Preemptive Job Scheduling with Priorities and Starvation Avoidance CUM Throughput Increasing Tool in Clusters

Balajee Maram[1]
[1] Asst. Prof., Dept of IT, GMR Institute of Technology,
Rajam – 532 127. balajee.m@gmrit.org , Cell No:08099557515.

Abstract: **This paper proposes a new scheduler to schedule parallel jobs on Clusters that may be part of a Computational Grid. This proposed policy proposes 3 Job Queues. In each Cluster, the first Queue has some jobs which are from Computational Grids. It means, it may be either bigger job or part of the bigger job from the Computational Grids. The second Queue has jobs which has low required execution time. The third Queue has jobs which has high required execution time. In first and second Queues, there is no chance of starvation. But in third Queue, there is a chance of starvation. So this proposed policy applied AGING technique to preempt the jobs which has low priority. And the first Queue is fully dedicated to execute a part of bigger jobs (Meta-Jobs) only. So here we maintain three job Queues which are effectively separate jobs according to their required execution time for local Jobs, bigger job/part of bigger job(Meta-Job) from Computational Grids. Here we preempt jobs by applying AGING Technique. Initially 20% of total available resources (processors) will be allocated to first Queue only. And 40% of total available resources (processors) will be allocated to second and third queues respectively. Whenever the third queue is Empty then the proposed scheduler simply selects the job which has least required execution time and executes it immediately. In this way, the proposed scheduler increases the throughput in clusters.**

Keywords: **Computational Grid, Parallel Systems, Preemption, Aging, Job Queues, required execution time.**

I. INTRODUCTION

In a single system, it is very easy to manage different types of resources (processors) and scheduling the resources (processors). But in the case of Grid, Resource management and job scheduling still is one of the challenging problems. Because there is a need of human communication on all levels is necessary to partition the application, locate resources (processors), and observe the behavior of distributed modules. One of the major challenges includes co-scheduling distributed applications across multiple independent systems, each of which may itself be parallel with its own scheduler. Using the required execution time, the scheduler rearranges the waiting queue.

In Clusters, scheduling jobs imposes additional challenges. The main objective of the proposed policy must serve to two classes of jobs: local jobs (sequential or parallel) that should be executed in a timely manner, jobs external to Cluster (jobs that can be executed according to agreed Time Slot). In this paper, we propose a new policy that is Preemptive job scheduling with Aging by using Multiple-Job-Queues. And we can increase the throughput in clusters by allocating more resources (processors) to the waiting jobs.

II. EXISTING SYSTEMS

Previously so many schedulers are proposed for managing resources (processors) and scheduling different types of jobs. In FCFS, it doesn't create starvation but it will give more waiting time. In SJF, it will give less waiting time but it will create starvation. In PRIORITY SCHEDULING, it will create starvation.

In previous work, a multiple-queue aggressive backfilling scheduling policy that continuously monitors system performance and changes its own parameters according to changes in the workload. But in previous work, there may occur starvation for bigger jobs which has low priority.

III. PROPOSED SYSTEM

A. *Scheduling Policies*

A non-FCFS scheduling policy that reduces fragmentation of system resources (processors) by permitting jobs to execute in an order different than their arrival. The goal of this multiple-queue policy is to reduce the likelihood of delaying a short job behind a long job. Within the context of scheduling resources (processors) in a computational grid, we supplement our multiple-queue preemption policy by considering static job priority levels and job reservations as follows. We consider jobs submitted by local users to have low priority and those jobs submitted from an external source (i.e., from elsewhere in the computational grid) to have high priority. Our goal is to serve these external jobs as quickly as possible without indicting delays on local jobs. We also assume that the system serves jobs that require execution at a specific time. Our goal is to accommodate these reservations as quickly as possible regardless of the consequences on remaining jobs. We now describe in detail the multiple-queue preemption policy with the necessary job prioritization and reservation schemes.

DOI: 01.IJCOM.01.03.521

B.*Multi-Queue preemptive Job Scheduling with Priorities*

Multiple-queue Preemptive Job Scheduling with AGING allows the scheduler to automatically change system parameters in response to workload fluctuations, thereby reducing the average job slowdown. Initially the scheduler collects the information about the available resources (processors) which includes in cluster. As shown in Fig:1, the proposed scheduler allocates 20% of total available resources(processors) to first queue of jobs. And in remaining 80% of resources (processors), 40% of available resources (processors) will be allocated to second queue and 40% of available resources (processors) will be allocated to third queue.

As time evolves, the partitions may exchange control of processors so that processors idle in one partition can be used for another job in another partition. But this method is applicable to second and third queues only. Therefore, partition boundaries become dynamic, allowing the system to adapt itself to changing workload conditions.

Because of AGING, the policy does not starve either a job that requires the entire machine for execution or a job with low priority. Sometimes the scheduler may change the workload of the nodes, and then sizes of Job-Queues may change into like in Fig1.

C**.** *Algorithm for selecting a job with low required execution time and high priority*

Step 1: if any processor is not idle which allocated to first queue then go to step 6?
Step 2: the proposed scheduler will check which job has least required execution time and high priority in third queue only.
Step 3: If job available, then the proposed scheduler discards that job from third queue and that job will be added to first queue.
Step 4: Now that job will be executed immediately.
Step 5: go to step 1
Step 6: continue with normal execution

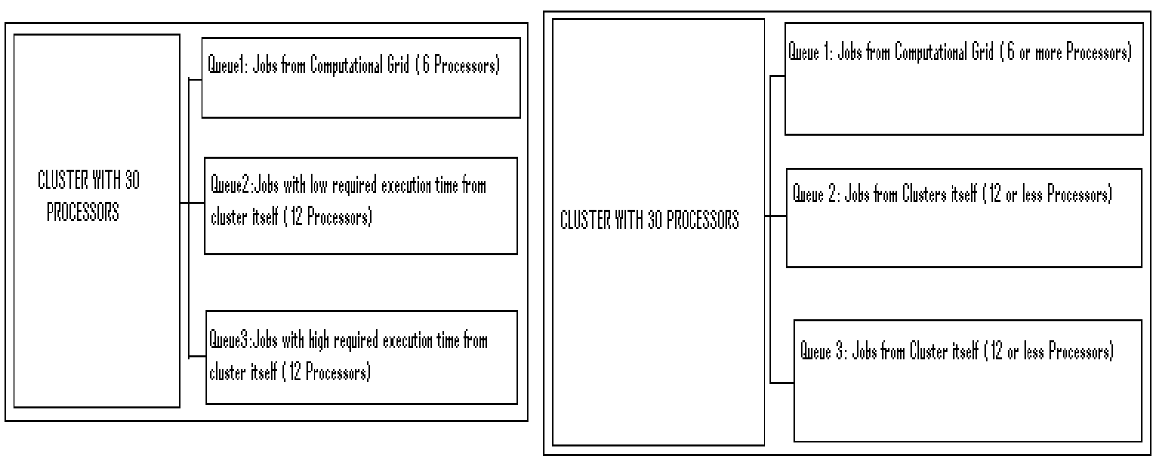


Fig 1: Initial partition boundaries and partition after load balancing in each Cluster

IV. EXPLANATION

When the user submits a bigger job to the Grid, then it divides the job into smaller pieces and allocates each piece (partition) of the Meta-Job to each Cluster. After agreed by the Clusters, that job will wait in first queue only.

When the job is from the cluster itself, then the classification of jobs in the following way. When a job is submitted, it is classified and assigned to the queue in partition p according to

$$p = \begin{cases} 2. & 0 < t_e < 3000 \\ 3. & 3000 \leq t_e \end{cases}$$

Where t_e is the required job execution time in seconds. If the arriving job cannot begin executing immediately, it is placed into the queue in partition p after all jobs of the same priority that arrived earlier. More specifically, if the job has high priority, it is placed into the queue after any high priority jobs that arrived before it. If the job has low priority, it is placed into the queue after all high priority jobs and after any low priority jobs that arrived before it.

When a big job is placed in the third queue then we are applying AGING technique to avoid starvation as well as to finish old big jobs early. In AGING, we can add some number '-n' to the existing priority. When that priority number reaches 1 then that job will be executed immediately.

DOI: 01.IJCOM.01.03.521

When a job, which is a part of bigger job (Meta-Job) comes, then it is placed in the first queue only. The newly coming job is placed in first queue according to their execution time slot. When the current time is equal to the previously agreed time for executing the part of the Meta-Job, then immediately it will be executed. If there is no sufficient nodes are available then immediately it stops some of the local jobs and allocate to the part of the Meta-Job. Otherwise it will take sufficient number of nodes and allocated to the part of Meta-Job. Then it will be executed immediately. So in this way it balances the low priority jobs, high priority jobs as well as part of Meta-Jobs.

Generally the first queue has meta-jobs or part-of meta-jobs. Now they are waiting for execution according to the previously agreed time slot. When any processor which allocated to first queue is idle, then the proposed scheduler find the job which has highest priority and least required execution time. If the proposed scheduler finds that type of job successfully, it simply discards that job from third queue. And that job will be added to the first queue. Then the proposed scheduler will execute that job immediately. It will continue this process when it finds the idle processors which allocated to the first queue only.

V. PERFORMANCE COMPARISON

The Proposed Scheduler performance can be done by using different examples. Here the comparisons between SJF, MQBS (Multiple-Queue Backfilling Scheduling) and Proposed Scheduler. Each Example has 8 jobs, RET and Priorities of those 8 jobs.

Example1

Job Name	P1	P2	P3	P4	P5	P6	P7	P8
RET	4	9	3	8	2	7	1	6
Priority	4	3	1	2	3	4	2	4

Example2

Job Name	P1	P2	P3	P4	P5	P6	P7	P8
RET	4	3	2	3	6	1	2	4
Priority	1	4	3	2	1	3	4	2

Example3

Job Name	P1	P2	P3	P4	P5	P6	P7	P8
RET	4	8	9	2	4	3	2	9
Priority	4	1	2	4	3	2	1	4

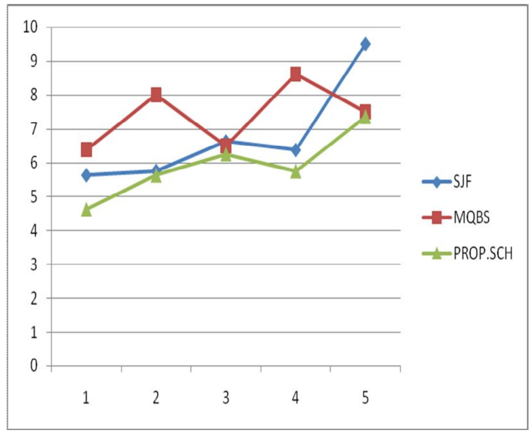
Example4

Job Name	P1	P2	P3	P4	P5	P6	P7	P8
RET	10		28	4	6	1	5	7
Priority	4	3	4	1	2	3	1	4

Example5

Job Name	P1	P2	P3	P4	P5	P6	P7	P8
RET	8	2	6	9	5	7	4	1
Priority	1	5	4	3	1	2	4	5

Overall Performance comparison of all examples (Ex1 to Ex5)



VI. CONCLUSION

Each job is assigned to a queue according to its required execution time. Each queue is assigned a non-overlapping partition of system resources(processors) on which jobs from the queue can execute. Partition boundaries change dynamically, adjusting to fluctuations in arrival intensities and workload mix. The multiple-queue policy significantly reduces the likelihood that a short job is overly delayed in the queue behind a very long job. And it is very useful for big jobs with low priority which entered into the queue early. By using this method, we can avoid starvation for bigger jobs. And we can increase the throughput by selecting and executing a job which has least required execution time and high priority.

VII. REFERENCES

1. Gred Quecke, Wolfgang Ziegler : MeSch – An Approach to Resource Management in a Distributed Environment.
2. Barry G. Lawson, Evgenia Smirni : Multiple-Queue Backfilling Scheduling with Priorities and Reservations for Parallel Systems.
3. Feitelson, D.G.: A Survey of Scheduling in Multiprogrammed Parallel Systems.Technical Report RC 19790, IBM Research Division.
4. IBM LoadLeveler: http://www.ibm.com/.
5. Keleher, P., Zotkin, D., and Perkovic, D.: Attacking the Bottlenecks in Backfilling Schedulers. Cluster Computing: The Journal of Networks, Software Tools and Applications. 3(4) (2000) 245{254.
6. Lawson, B.G., Smirni, E., and Puiu D.: Self-Adapting Backfilling Scheduling for Parallel Systems. Proceedings of the 2002 International Conference on

DOI: 01.IJCOM.01.03.521

Parallel Processing (ICPP 2002). August 2002, 583{592.

7. Maui Scheduler Open Cluster Software: http://mauischeduler.sourceforge.net/.
8. MÄualem, A. and Feitelson, D.G.: Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling. IEEE Transactions on Parallel and Distributed Systems. 12(6) (2001) 529{543.
9. ParallelWorkloadArchive: http://www.cs.huji.ac.il/labs/parallel/workload/.
10. Lifka, D.: The ANL/IBM SP scheduling system. Job Scheduling Strategies for Parallel Processing, Lect. Notes Comput. Sci. 949 (1995) 295–303
11. Grund, H., Link, P., Quecke, G., Ziegler, W. EASY Job Scheduler for Cenju-3. Proc. Of the First Cenju Workshop, HPC Asia '97, Seoul, Korea (1997) 20–34
12. Foster, I., Kesselman, C., (eds): The Grid: Blueprint for a Future Computing Infrastructure. Morgan-Kaufmann Publishers (1999)
13. Balajee Maram, B.Suresh, M.Suneetha, V.Vasudha Rani, G. Veerraju- "Preemptive job scheduling with priorities and starvation cum congetion avoidance in clusters" presented in IEEE conference on 9th-10th Feb, 2010.

VIII. VOTE OF THANKS

I am very thankful to the authors where I mentioned in Ref1&2.

DOI: 01.IJCOM.01.03.521

ACEEE